Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Novel Domains and Problems for Privacy-Preserving Multi-Agent Planning

Victoria Shmatova

Supervisor: Ing. Antonín Komenda, Ph.D.
Field of study: Computer and Information Science
May 2018

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Shmatova Victoria**	Personal ID number:	**434726**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Branch of study:	**Computer and Information Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Novel Domains and Problems for Privacy-Preserving Multi-Agent Planning

Bachelor’s thesis title in Czech:

Nové domény a problémy pro multi-agentní plánování s privátní znalostí

Guidelines:

The thesis will provide design, formalization and experimental analysis of new planning domains and problems for domain-independent privacy preserving multi-agent planning. Such domains are needed in the multi-agent planning community for benchmarking, competitions and testing and comparison of the existing and developed planners. The student will propose novel domains based on existing puzzles, video games, and real-world problems. The domains will be tested on a set of selected existing multi-agent planners.
1) Study literature in the area of multi-agent planning, especially the privacy-preserving variants.
2) Analyze existing domains and problems used for privacy-preserving multi-agent planning, find properties hard from the planning and privacy preservation point of view, not covered by the existing domains/problems.
3) Propose novel planning domains and problems for privacy-preserving multi-agent planning (some of the domains and problems should target the properties from 2)
4) Formalize the proposed domains and problems in a language used for multi-agent planning (e.g., MA-PDDL).
5) Experimentally compare selected existing multi-agent planners on the proposed domains/problems and analyze their behavior.

Bibliography / sources:

[1] Malik Ghallab, Dana S. Nau, Paolo Traverso: Automated Planning: Theory & Practice. Morgan Kaufmann 2004.
[2] Stefan Edelkamp, Stefan Schroedl: Heuristic Search: Theory and Applications. Morgan Kaufmann. 2012.
[3] Raz Nissim, Ronen I. Brafman: Distributed Heuristic Forward Search for Multi-agent Planning. J. Artif. Intell. Res. 51: 293-332 (2014).
[4] Michal Štolba, Antonín Komenda: The MADLA planner: Multi-agent planning by combination of distributed and local heuristic search. Artif. Intell. 252: 175-210 (2017).
[5] Jan Tožička, Michal Štolba, Antonín Komenda: The Limits of Strong Privacy Preserving Multi-Agent Planning. ICAPS 2017: 297-305.

Name and workplace of bachelor’s thesis supervisor:

Ing. Antonín Komenda, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **18.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2018**

Ing. Antonín Komenda, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor’s signature	Head of department’s signature	Dean’s signature

CVUT-CZ-ZBP-2015.1 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

I would like to thank my supervisor Ing. Antonín Komenda, Ph.D. for his invaluable help and guidance throughout the writing of this thesis.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instruction for observing the ethical principles in the preparation of university theses.

Prague, 24 May 2018

Abstract

The objective of this thesis is to propose, formalize and implement new planning domains and problems for domain-independent privacy-preserving multi-agent planning. The work focuses on formalizing and implementing multi-agent planning problems by using multi-agent extension of STRIPS model and multi-agent extension of PDDL with privacy. The proposed domains will complement the small set of already existing multi-agent domains and be used by multi-agent planning community for benchmarking, testing and possible future multi-agent planning competitions.

Keywords: planning, multi-agent planning, MA-PDDL

Supervisor: Ing. Antonín Komenda, Ph.D.

Abstrakt

Cílem této bakalářské práce je navrhnout, formalizovat a implementovat nové domény a problémy pro multi-agentní plánování s privátní znalostí. Práce se zaměřuje na formalizaci a implementaci multi-agentních plánovacích problémů pomocí multi-agentního rozšíření STRIPS modelu a multi-agentního rozšíření PDDL s privátní znalostí. Navrhované domény doplní již existující domény pro multi-agentní plánování a mohou být využity multi-agentní plánováci komunitou pro benchmarking, testování a případné budoucí plánovací soutěže.

Klíčová slova: plánování, multi-agentní plánování, MA-PDDL

Překlad názvu: Nové domény a problémy pro multi-agentní plánování s privátní znalostí

Contents

Figures

Tables

Chapter 1

Introduction

The objective of this work is to provide new domains and problems that will be able to adequately assess the efficiency of domain-independent privacy-preserving multi-agent planners. The work focuses on those issues that are not sufficiently represented in the existing domains for multi-agent planning and may be challenging for privacy-preserving planners.

There already exists a set of domains and problems that was used in the Competition of Distributed and Multiagent Planners (CoDMAP) [KvK16], but most of those domains and problems were initially designed for classical single-agent planning and some of them do not precisely exploit the features provided by multi-agent privacy-preserving planning. From this comes the need to come up with new domains and problems that the multi-agent planning community will be able to use for competitions, testing, and comparison of multi-agent planners.

The work is structured as follows. First, we provide sufficient background information on automated planning and describe the tools used to formalize single-agent problems. Then we extend it to work with multi-agent problems, discuss issues of privacy in multi-agent planning, and give a brief description of CoDMAP and some of the existing multi-agent planners. Then we propose, formalize and test new domains and problems on selected multi-agent planners and summarize the results.

This work will produce the implementation of novel domains and provide sets of diverse problems to be used as benchmarks.

Part I

From Automated to Multi-agent Planning

Chapter 2

Laying The Foundation

2.1 Automated planning

Automated planning is a branch of Artificial Intelligence that studies the process of choosing and organizing actions for changing the state of a system by anticipating their expected outcomes [NGT04].

There are two main approaches to planning:

Domain-specific approach uses techniques that are specifically designed or tuned for a particular domain and are limited to being able to only work with it. Although domain-specific planners can perform quite well on a specific set of problems, they are generally expensive and time-consuming to build, with questions thus concerning overall efficiency, given that the planners are dedicated to doing just one thing.

Domain-independent approach, on the other hand, relies on abstract and general models. Domain-independent planners have no specific knowledge about the domain apart from the information provided in problem description and can solve a wild range of problems. As a downside, this flexibility of domain-independent planners may, at times, come at the cost of lower performance.

For the most part, automated planning is drawn to determining a general way to solve problems, but it is important to note that it is not opposed to domain-specific planning [NGT04]. Domain-independent techniques can serve as a foundation for domain-specific tasks and domain-specific techniques may sometimes bring something new to domain-independent planning. Nevertheless, from here on we will focus on domain-independent planning.

The essence of automated planning is in defining problems explicitly and finding solutions to them automatically. This work concentrates on planning from the perspective of formulating and representing problems, leaving out the underlying mechanisms of solving the problem.

2.2 Problems in Classical Planning

This section provides standard tools for formalization and implementation of problems in classical planning. This will serve as a foundation for working with multi-agent problems.

2.2.1 Formulation of a classical planning problem

Here we consider a finite system with deterministic non-durative actions and complete information.

A planning problem can be represented as a state model T with:

- a finite state space S,
- an initial state s_0,
- a set of goal states $S_G \subseteq S$,
- a finite set of actions A,
- a transition function $\gamma(s, a) = s'$, where $s, s' \in S$ and $a \in A$, and
- a non-negative cost function $C(a)$, $a \in A$.

A solution (plan) to a planning problem is a sequence of actions $(a_0, ..., a_n)$ corresponding to a sequence of transitions $(s_0, ..., s_{n+1})$, where $s_{n+1} \in S_G$ and $\gamma(s_k, a_k) = s_{k+1}$. The optimal solution to a problem is a sequence of actions $(a_0, ..., a_n)$ with minimal $\sum_{i=0}^{n} C(a_i)$.

Following this, we can define a planning domain as a restricted state-transition system $\Sigma = (S, A, \gamma)$.

2.2.2 STRIPS representation

In planning, problems are defined implicitly using a declarative language. One of the most popular ways of representing a problem is by using STRIPS notation. STRIPS is a formal language that allows one to represent a planning problem as [BG00]:

- a finite set of atoms (also called propositions) P, where states $s \in S$ are collections of atoms from P, i.e. $s \subseteq P$,
- a state $s_0 \subseteq P$, that represents the initial state of the system,
- a set $S_G \subseteq P$, that represents the conditions required to reach a goal state, and

- a finite set of actions A.

Actions are the driving force of the STRIPS paradigm. Each action $a \in A$ has:

preconditions pre(a), a set of atoms that needs to be true for the action to be applicable.

effects that describe the changes in the system after applying the action a:

Del(a), a set of atoms deleted from the state.

Add(a), a set of atoms added to the state.

In other words, to apply an action $a \in A$, pre(a) have to be a subset of the current state of the system and the result of it is a new state $s' = (s \backslash Del(a)) \cup Add(a)$ [vO09].

2.2.3 PDDL

PDDL (Planning Domain Definition Language) is a language used in a majority of planning systems. It is a standard language of the International Planning Competition (IPC) [VCM18] and many planners support one or another version of PDDL (which includes STRIPS and various extensions).

In PDDL the problem is divided into two parts: domain description and problem description. Domain description contains specific information about a domain, such as the properties of objects (in form of predicates) and the definition of possible actions. The problem description contains information about a particular problem: the objects, initial state, and the specification of goal states.

Domain description

PDDL is a modular language. It consists of subsets of features (modules) called requirements. Used modules have to be explicitly specified for every domain in the :requirements part of the domain definition.

Every domain definition starts with `(define (domain <domain name>)...)`. Then it is followed by the specification of requirements for this domain. `:strips` requirement means that the domain uses the STRIPS subset of the language. `:typing` means that the object types are used. The type hierarchy is specified at `:types`) in the form of `<list of types> - <parent type>`, where a type declared in `<list of types>` is either a final type or a parent type for another `<list of types>` (see Listing 2.1, where `movable` and `location` extend the build-in supertype `object`; and `thing` and `another-thing` extend `movable`).

The properties of objects and the relations between objects (atomic formulas) are defined in the `:predicates` field. Each predicate has the form `(<predicate name> <argument 1> ... <argument n>)`, where an argument defines a parameter that starts with a question mark and is followed by its type (e.g., `?t - thing`). This part only specifies names and arguments of the predicates, while the predicates themselves are listed in the problem description.

Listing 2.1: An example of a simple domain definition.

```
(define (domain simple-world)
  (:requirements :strips :typing)
  (:types movable location - object
          thing another-thing - movable)
  (:predicates
    (at ?t - movable ?l - location)
    (clear ?l - location)
    ...
   )
   ...
)
```

The actions are defined with (i) the `:parameters` field that specifies arguments and their types, (ii) the `:precondition` field that contains an atom or a conjunction of atoms that need to be true to allow application of the action, and (iii) the `:effect` field that describes the result of applying the action, which can be an added atom of the format `(<predicate name> <argument 1> ... <argument n>)`, a deleted atom of the format `(not (<predicate name> <argument 1> ... <argument n>))` or a conjunction of added and deleted atoms (see Listing 2.2).

Listing 2.2: An example of an action in the simple-world domain. The action describes moving the "thing" object from the ?from location to the ?to location.

```
(:action move-thing
    :parameters (?t - thing ?from - location
                 ?to - location)
    :precondition (and (at ?t ?from)
        (clear ?to))
    :effect     (and (not (at ?t ?from))
        (not (clear ?to))
        (at ?t ?to)
        (clear ?from))
)
```

Problem description

The problem description contains information about a particular problem. It always starts with `(define (problem <problem name>)...` and follows with the specification of the domain (`:domain <domain name>`). The object instances and their types are then listed in the `:objects` section in the format `<object name> - <type>` (see Listing 2.3).

The `:init` and `:goal` sections depict the initial and goal state respectively with the predicates in the format of `(<predicate name> <object name 1> ... <object name n>)`.

Listing 2.3: An example of a simple problem definition.

```
(define (problem simple-problem)
  (:domain simple-world)
  (:objects
    simple-thing - thing
    location-1 - location
    location-2 - location
    ...)
  (:init
    (at simple-thing location-1)
    (clear location-2)
    ...)
  (:goal
    (at simple-thing location-2)
    )
)
```

2.3 Summary

To summarize, in this chapter we described a formal way of working with single-agent planning problems. Next, we will expand on this system with the notion of multiple planning agents.

Chapter 3

Introducing a new dimension to planning

3.1 Multi-agent planning

Multi-agent planning (MAP) is a relatively recent field of study that introduces a new perspective on automated planning. Instead of treating a problem as a single-agent task, multi-agent planning employs multiple intelligent agents that share a common goal. Knowledge about the world and action capabilities may be distributed among the agents, but the objective is still to find a solution that leads from the initial state to the specified goal.

Such distributed planning can be divided into:

- cooperative and distributed MAP that focuses on formulating a plan among multiple agents in the distributed environment; and
- decentralized planning by multiple agents for multiple agents in a shared environment [TnOKv17].

It is also worth distinguishing centralized planning for multiple agents. The 2015 Competition of Distributed and Multiagent Planners (CoDMAP) [KvK16] included both Decentralised and Centralised tracks, but the latter was viewed as a “transitional” track toward "pure" multi-agent planning [cod15].

Here we will concentrate on domain-independent deterministic cooperative multi-agent planning, which concerns the collective effort of multiple agents in finding a solution to the problem that the agents could not have solved by themselves (as efficiently or at all) [Dur01]. Here the planning agents are considered not to be motivated by their own advantage, hence they work together on reaching a common goal.

3.2 Multi-Agent Extension of STRIPS

The majority of multi-agent planners to some extent adopt MA-STRIPS [BD08] formalism, the minimal extension of STRIPS for multi-agent planning, which is by far the most common way of describing problems in the multi-agent community. MA-STRIPS extends STRIPS with two notions [vKK16]: factorization that defines what actions can be performed by which agents; and privacy that defines the scope of atoms and actions.

A MA-STRIPS problem [BD08] for a set of agents $\Phi = \{\phi_i\}_{i=1}^{n}$ is given by a 4-tuple $= <P, \{A_i\}_{i=1}^{n}, s_0, S_G>$, where:

- P is a finite set of atoms,
- $s_0 \subseteq P$ is the initial state,
- $S_G \subseteq P$ is the goal state condition,
- for $i \in \{1, ..., k\}$, A_i is the set of actions of the agent ϕ_i. Each action is defined, as in standard STRIPS, by its preconditions and effects. With $k = 1$ the problem is reduced to a standard single-agent problem.

A solution to the planning problem is a sequence of actions, each of which is associated with an agent that performs it, which leads from the initial state to the state that satisfies the task goals G.

Privacy comes from factorization by dividing actions into private and public. An atom $p \in P$ is private to an agent ϕ_i if it is only used by this agent and only affected by the actions that the agent ϕ_i can perform and no other agent has any knowledge about it, otherwise the atom p is public to all agents in Φ. In cooperative planning, the goal G is assumed to be shared among all the agents and therefore is always public.

The set of all public atoms and the set of private atoms of any agent ϕ_i are disjoint. Together all public atoms and all sets of private atoms of the agents form the entire P.

Similarly, an action $a \in A_i$ is private if it only contains and affects atoms private to the agent ϕ_i, otherwise the action is public. When a public action is executed, all agents are aware of it and its public effects, while execution of private actions is not revealed to other agents.

This extension of STRIPS is the bare minimum required to properly define a multi-agent task. The definition of privacy here is rather strict and it does not allow joint actions between multiple agents, but it is still a solid foundation for formalizing the multi-agent model.

3.3 Multi-Agent Extension of PDDL

With a multi-agent extension of STRIPS comes a multi-agent extension of PDDL. MA-PDDL was first proposed by Daniel L. Kovacs in [Kov] and later was developed in line with MA-STRIPS for the 2015 CoDMAP competition as an attempt to standardize problem specifications for multi-agent planners in order to be able to test them on a common set of problems.

The CoDMAP version of MA-PDDL includes two main features of MA-STRIPS: privacy and factorization [vKK16], and allows one to define agents as world objects. There are two specifications of MA-PDDL:

- **unfactored specification** (`:unfactored-privacy` requirement) that uses a single domain and problem description; and
- **factored specification** (`:factored-privacy` requirement) that requires a separate domain and problem description for each planning agent.

3.3.1 Privacy

The definition of MA-PDDL privacy (for both factored and unfactored specifications of privacy) complies with the following rules [vKK16]:

- That a public predicate grounded with public objects (a grounded predicate means a predicate that has all parameters replaced with world objects) is a public fact;
- That a public predicate grounded with at least one private object of an agent is a private fact of this agent; and
- That a predicate private to an agent is always grounded to a private fact of this agent.

Unfactored privacy

Similarly to the PDDL description of the problem, in MA-PDDL the problem is divided into a domain description that contains information specific to a particular domain and a problem description with specifications of a particular task.

To indicate that the problem uses unfactored privacy specifications, the `:unfactored-privacy` requirement is specified (see Listing 3.1), as well as the `:multi-agent` requirement for multi-agent planning.

In MA-STRIPS the agents are not defined as a part of the problem and factorization is accomplished through the sets of actions that are associated with different agents. In the unfactored version of MA-PDDL, as all information about the problem is specified in one place, it is logical to treat agents as objects, for it is a convenient way to associate them with predicates and actions.

The `(:private <agent> ...)` expression in the definition of predicates denotes the predicates private to the specified agent.

Listing 3.1: An excerpt of domain definition from the unfactored version of the depot domain used in the CoDMAP 2015 competition [KvK16].

```
(define (domain depot)
    (:requirements :typing :multi-agent
       :unfactored-privacy)
(:types
    place locatable driver - object
    depot distributor - place
    truck hoist surface - locatable
    pallet crate - surface
)
(:predicates
    (at ?x - locatable ?y - place)
    (on ?x - crate ?y - surface)
    (in ?x - crate ?y - truck)
    (clear ?x - surface)

    (:private ?agent - place
       (lifting ?agent - place ?x - hoist ?y - crate)
       (available ?agent - place ?x - hoist)
    )

    (:private ?agent - driver
       (driving ?agent - driver ?t - truck)
    )
)
...
```

The definition of actions is similar to regular PDDL with the addition of the `:agent <agent>` field that specifies the agent that the action belongs to (e.g., `:agent ?a - driver`, where an object of the type driver is an agent that can perform the action) (see Listing 3.2). There are no separate declarations of private and public actions as this division unambiguously follows from the definition of privacy.

Listing 3.2: An excerpt of action definition from the unfactored version of the depot domain used in the CoDMAP 2015 competition [KvK16].

```
(:action drive
    :agent ?a - driver
    :parameters (?x - truck ?y - place ?z - place)
    :precondition (and (at ?x ?y)
        (driving ?a ?x))
    :effect (and
        (at ?x ?z)
        (not (at ?x ?y)))
)
```

In the problem description, the information private to an agent should also be enclosed in `(:private <agent name> ...)` expression (see Listing 3.3). It is recommended that the objects representing agents should be private [KvK16], otherwise agents of the same type would be able to ground and use each other's actions.

Listing 3.3: An excerpt from the unfactored version of the depot problem definition used in the CoDMAP 2015 competition [KvK16].

```
(define (problem depotprob1818)
(:domain depot)
(:objects
    truck1 - truck
    truck0 - truck
    ...
    (:private driver1
        driver1 - driver
    )
    (:private driver0
        driver0 - driver
    )
)
...
```

Factored privacy

In factored MA-PDDL, each planning agent is provided with a separate domain and problem description that specifies information only relevant to the particular agent [FK18], which includes public predicates, private predicates of the agent and agent's actions. According to this scheme, grounded predicates and instances of the objects that are public, and thus are shared among the agents, are denoted with the same names in every description file.

To indicate that the problem uses factored privacy specifications, the `:factored-privacy` requirement is specified (see Listing 3.4).

In the domain description of the agent, the `(:private ...)` expression in the :predicates section denotes the predicates private to the agent. In the factored version of the language, it is not necessary to specify agents in the `:private` field, as the scope of each agent is defined within a separate file. The actions that the agent is capable of performing are defined as in regular PDDL.

Listing 3.4: Domain description for a driver in the factored version of the depot domain used in the CoDMAP 2015 competition [KvK16].

```
(define (domain depot)
(:requirements :factored-privacy :typing)
(:types
     place locatable driver - object
     depot distributor - place
     truck hoist surface - locatable
     pallet crate - surface
 )
(:predicates
    (at ?x - locatable ?y - place)
    (on ?x - crate ?y - surface)
    (in ?x - crate ?y - truck)
    (clear ?x - surface)

    (:private
        (driving ?agent - driver ?t - truck)
    )
)
...
```

In the problem description, the information private to the agent should also be enclosed in the `(:private ...)` expression in the :objects section (see Listing 3.5).

Listing 3.5: Problem description for a driver in the factored version of the depot domain used in the CoDMAP 2015 competition [KvK16].

```
(define (problem depotprob1818)
(:domain depot)
(:objects
    truck1 - truck
    ...
    (:private
        driver0 - driver
    )
)
...
```

3.3.2 A bit more about privacy

Privacy is one of the keystones of multi-agent planning. There are many problems with sensitive information that planning agents for various reasons cannot or do not wish to disclose. How to treat such information is generally up to planners.

Some planners ignore the issue of privacy and provide no special treatment of private information.

Privacy-preserving planning [NB14] aims not to disclose private information. In the weak preservation of privacy, agents do not explicitly share any private information (by the means of encrypting private information or sharing only public "interfaces" of actions). Nevertheless some information can be deduced by the agents during the execution time (e.g., by the agents of the same "type"). In short, weak privacy preservation can be easily achievable, but it is not considered secure [TnOKv17].

Preservation of privacy is considered to be strong if no agent can obtain any information private to another agent from the public information available during the execution time. At the moment, this matter is in literature more theoretical than practical.

3.4 Summary

It is still in the early days of multi-agent planning and although the privacy defined in MA-PDDL allows a certain degree of freedom in describing problems [vKK16], many things can be added to its functionality, such as support of facts and objects private to a group of agents or support of joint actions between agents. On the whole, the tools described in this chapter are more than sufficient to provide us with a formal way to work with multi-agent problems.

Chapter 4

Competition of Distributed and Multiagent Planners

The Competition of Distributed and Multiagent Planners (CoDMAP) [KvK16] was a part of the 3rd Workshop on Distributed and Multi-Agent Planning (DMAP) 2015. The competition was organized by Michal Štolba, Antonín Komenda, and Daniel L. Kovacs and was regarded as "a preliminary version of a possible future DMAP track at the International Planning Competition (IPC) and focused on comparing domain-independent, offline planners for multiple cooperative agents" [cod15]. The purpose of this competition was to provide a common ground for the comparison of multi-agent planners [KvK16]. It was agreed that the prerequisites for planners to enter the competition had to be compatibility with the MA-STRIPS model and support MA-PDDL as the input language.

4.1 Competition Tracks and Benchmarks

The CoDMAP competition had two tracks: centralized and distributed, without separation to satisficing and optimal (the goal of optimal planning is to produce a plan with minimal cost, while in satisficing planning any plan is acceptable, although less expensive plans are preferred). The Centralized Track was run on a single multi-core machine. The memory space was shared and no restrictions were placed on the means or form of communication between agents [KvK16]. Planners were allowed to accept the unfactored as well as the factored version of MA-PDDL and MA-STRIPS requirements in this track were relaxed. The output in this track had to be a valid linear plan. Such restrictions allowed a large number of planners to participate. The Distributed Track was more strict, it was run on several independent machines - one machine per planning agent with its own description of the domain and problem (thus only factored MA-PDDL) with communication over a TCP/IP network and without sharing any private information. The output had to be a set of linear plans that each agent could execute in parallel [cod15]. This track would serve as a prototype for the potential multi-agent IPC track (IPC, the International Planning Competition [VCM18] is an event organized within the International Conference on Planning and Scheduling (ICAPS) [BFMS17] to empirically compare "state-of-the-art" planning systems).

The benchmarks for the competition were comprised of ten multi-agent versions of classical IPC domains and only two novel domains designed specifically for multi-agent planning. From this comes the need to devise new domains for multi-agent planning, as the planning community would benefit from these and problems to test and compare the capabilities of multi-agent planners.

Chapter 5

Multi-agent Planners

In this section we provide brief descriptions of the planners that will later be tested on proposed domains and problems.

5.1 MAPlan

MAPlan (Daniel Fišer, Michal Štolba and Antonín Komenda) [FvK15] is a complete (can be satisficing and optimal) multi-agent planner "based on ideas introduced by the MAD-A* planner" (a distributed version of A*). It supports multi-threaded as well as distributed planning and accepts both factored and unfactored versions of MA-PDDL.

The planner employs weak privacy preservation and has altogether four different heuristics, two of which are tested in this work: (i) the lazy variant of distributed Fast-Forward (FF) heuristic [vK14], a heuristics based on finding a plan to a relaxed problem, and (ii) the landmark cut (LM-cut) heuristic [HD09], an admissible heuristic that iteratively computes the estimate value from landmarks (landmark is a set of actions such that each plan for a problem has to contain at least one of them) and their cost functions.

5.2 MADLA

MADLA (Michal Štolba and Antonín Komenda) [vK15] is a (satisficing) multi-agent planner based on the greedy best-first search algorithm. The planner runs in multiple threads (with separate thread for each agent) with the agent communicating "either using in-process communication or via TCP-IP local loopback". The planner accepts either plain PDDL with a list of agents (ADDL) or an unfactored version of MA-PDDL that is then translated into plain PDDL + ADDL.

The planner employs weak privacy preservation and combines two variants of Fast-Forward heuristic: (i) projected, where the heuristic is estimated locally for the part of the problem related to a particular agent, and (ii) distributed (variant of the lazy Fast-Forward heuristic [vK14]), where a global heuristic estimate is calculated by the means of coordinated computation of multiple agents [vK17].

Part II

Novel domains and problems

Chapter 6

Domains and Problems

In this chapter we propose novel domains and problems inspired by existing computer games and real world problems. The aim is to present and formalize properties that are currently underrepresented in the benchmarks for privacy-preserving multi-agent planning.

6.1 Lanterns

We decided to start with a multi-agent domain that puts emphasis on co-ordination and cooperation between agents in its purest form. The first domain was inspired by the two-player co-op puzzle TRYST developed by Ryan Welham, Mike Gillespie, and Sean Leach [RW17]. The idea of the game is to traverse a world obstructed with objects that can be distantly controlled by the players. Getting through this world requires constant cooperation and is impossible for the players to do by themselves, which is why it was a perfect candidate for the first domain.

In this and following chapters, we will be using only the factored version of MA-PDDL as it is a preferred version for distributed multi-agent planning.

We would also like to note that the initial game contains a lot of details that serve well in engaging a human player, but are not necessary for solving the puzzle automatically as they would certainly overcomplicate the implementation. In this work we aim for as simple and elegant solution as bulky a language as MA-PDDL would allow, thus we decided to lean in the direction of a much-simplified version of the puzzle.

6.1.1 Description

The lanterns domain is about multiple players traversing the world composed of rooms and locked/unlocked doors. Their goal is to collect and light all the lanterns scattered around.

6.1.2 Formalization

The objects in our implementation can be divided into two categories: (i) the objects that describe the layouts of the world (`door`, `window`, `room`, and `switch`) and (ii) movable objects (`player`, `lantern`) that can change their position throughout the course of the game. The predicate `(in ?o - thing ?r - room)` specifies the position of the movable objects.

The world of the game consists of rooms connected to one another via doors. A player can move from one room to the room adjacent to it. There also has to be an unlocked `(unlocked ?d - door)` door connecting two rooms. The fact that the two rooms are adjacent is marked with the static predicate (static predicates are not changed by actions) `(adjacent-rooms ?r1 - room ?r2 - room ?d - door)`.

Locked `(locked ?d - door)` doors can be unlocked distantly with a switch located anywhere in the world. The static predicate `(switch-to-door ?sw - door ?s - switch)` specifies which switch unlocks which door. For simplicity, we decided to avoid the situation in which several doors can be open with one switch, as this one-to-many situation can be easily translated into a similar problem with several one-to-one switches located in one place. The location of the switch is noted with the static predicate `(adjacent-switch ?r - room ?s - switch)`. In order to unlock a door, a player has to stand on the switch to that door. While the predicate `(on ?p - player ?s - switch)` is true, the door is unlocked (see Listing 6.1).

To illustrate this, let's consider the following situation: if one player wants to get from room a to room b and there is a door (locked with switch s) between rooms a and b, the order in which things happen is as follows:

1. first player is in room a,
2. second player presses switch s,
3. corresponding door opens,
4. first player moves to room b,
5. second player releases the switch,
6. door closes.

There are also windows between rooms that allow players to pass lanterns without the need to unlock the door between rooms. The fact that there is a window between two rooms is denoted with the static predicate `(window ?r1 - room ?r2 - room)`. Lanterns can be either lit `(lighted ?l - lantern)` or not lit `(not-lighted ?l - lantern)`. In order to light a lantern, the lantern has to be in the room with fire `(room-with-light ?r - room)`.

The goal is to light all the lanterns (`lighted ?l - lantern`) present in the problem. Players traverse the world, collect lanterns and bring them all to the room with a fire to light them. A player holding a lantern is denoted using the (`holding-lantern ?p - player ?l - lantern`) predicate. A player without a lantern is denoted using the (`without-lantern ?p - player`) predicate.

Listing 6.1: An excerpt from the lanterns domain action definition.

```
; an action describing stepping on a switch
(:action move-room-to-switch
    :parameters (?p - player ?r - room ?s - switch
                 ?d - door)
    :precondition (and
        (in ?p ?r)
        (adjacent-switch ?r ?s)
        (switch-to-door ?d ?s)
        (locked ?d)
    )
    :effect (and
        (not (in ?p ?r))
        (on ?p ?s)
        (unlocked ?d)
        (not (locked ?d))
    )
)
; an action describing releasing a switch
(:action move-switch-to-room
    :parameters (?p - player ?r - room ?s - switch
                 ?d - door)
    :precondition (and
        (on ?p ?s)
        (adjacent-switch ?r ?s)
        (switch-to-door ?d ?s)
        (unlocked ?d)
    )
    :effect (and
        (not (on ?p ?s))
        (in ?p ?r)
        (locked ?d)
        (not (unlocked ?d))
    )
)
```

6.1.3 Problems

For this domain we designed a set of problems with various configurations of players, lanterns and door-switches (initial states of some of these problems are shown in Figure 6.1, where each switch corresponds to a door of the same color) to see how different changes in the description of the problems affect the ability of planners to solve them.

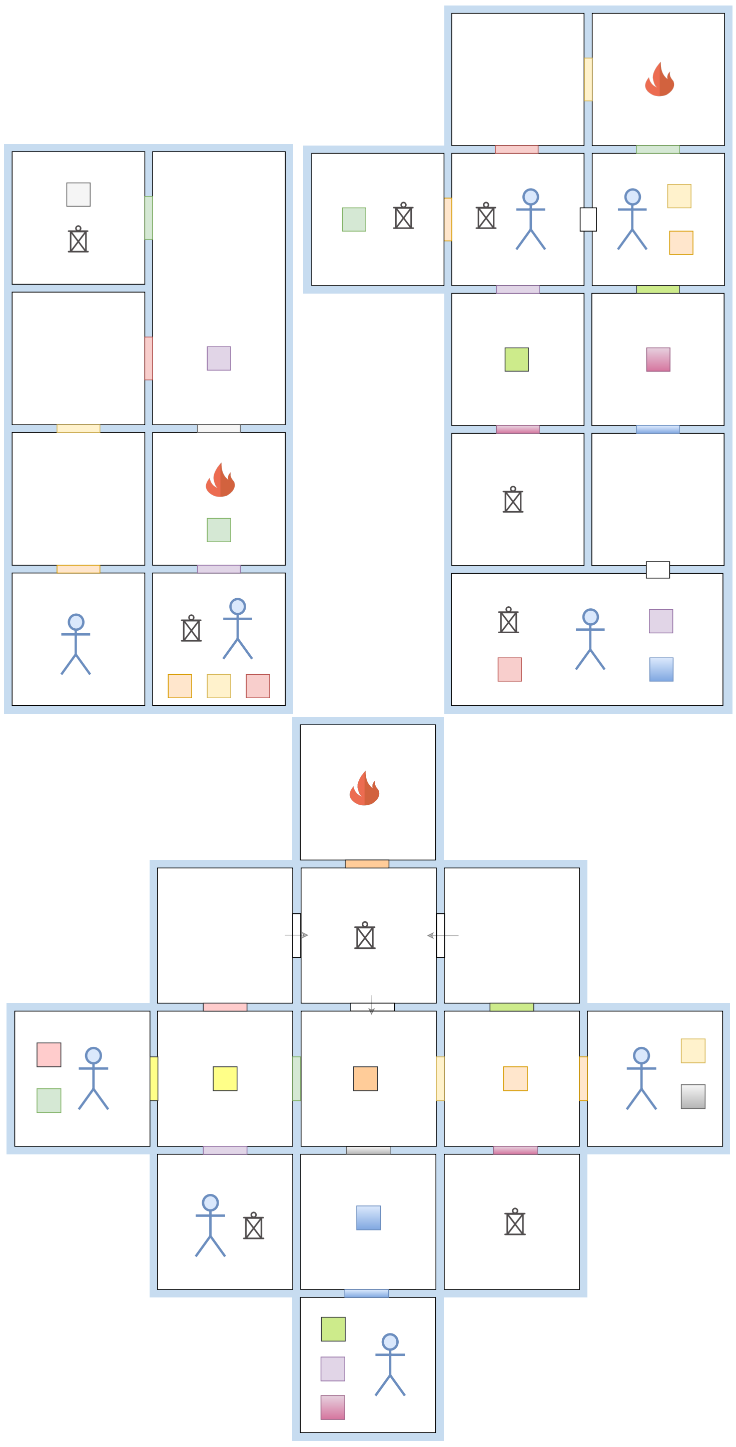

Figure 6.1: Illustration of the problems from the lanterns domain.

6.2 Hierarchical Logistics

The second domain concentrates on the issue of structural ordering in problems, which is currently underrepresented in the set of benchmarks for multi-agent planners. This domain is a hierarchical variation of the classical logistics domain.

6.2.1 Description

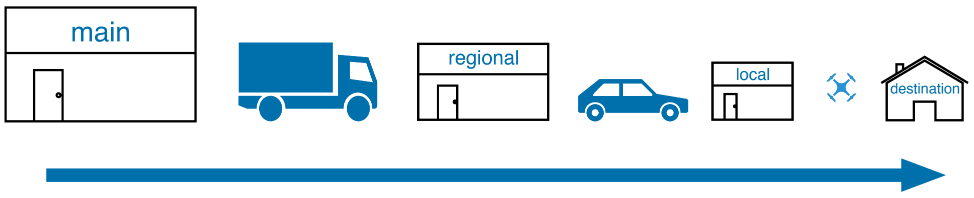

Figure 6.2: Illustration of the problem from the hierarchical logistics domain.

In this domain, the objective is to deliver packages from the main warehouse to their destinations. Each package goes through the route from the main warehouse to a regional warehouse to a local warehouse and finally to its destination (see Figure 6.2). Each warehouse (main, regional, local) has vehicles assigned to it that deliver packages one level down the line. From the main warehouse, the packages are sent on a truck to a regional warehouse, up to three packages at a time. From the regional warehouse, the packages go via a car to a local warehouse, up to two packages at a time. From there the packages are delivered by drones, one at a time.

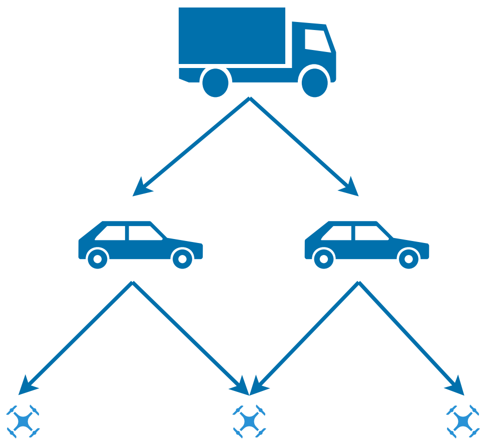

Figure 6.3: Interactions between agents in the hierarchical logistics domain.

The agents of the hierarchical logistics domain are the vehicles (trucks, cars, and drones) that deliver packages from one location to another. The relations between the agents can be represented with the agent interaction digraph (see Figure 6.3) introduced in [BD08], where a directed edge from agent ϕ_i to agent ϕ_j shows that there exists action $a_1 \in A_i$ that can affect the action $a_2 \in A_j$. In this particular case, agent ϕ_i provides the condition needed for the agent ϕ_j to execute its actions.

6.2.2 Formalization

The objects in the implementation can be divided into three categories: (i) the objects that represent the vehicles (`truck`, `car`, and `drone`), (ii) the objects that represent locations (`destination` and `warehouse`), and (iii) the packages (`package`) that need to be delivered (see Listing 6.2).

Listing 6.2: An excerpt from the hierarchical logistics domain definition.

```
(define (domain hierarchical-logistics)
(:requirements :strips :factored-privacy :typing)
(:types
     location vehicle package - object
     warehouse destination - location
     truck car drone - vehicle
 )
(:predicates
    ; information about the world
    (map ?from - warehouse ?to - location)
    ; information about vehicles
    (at ?m - package ?r - location)
    (address ?p - package ?d - destination)
    (delivered ?p - package)
    ; information about vehicles
    (:private
        (depo ?v - vehicle ?w - warehouse)
        (delivering ?v - vehicle)
        (at-depo ?v - vehicle)
    )
)
```

The structure of the world in the problem is described with the static `(map ?from - warehouse ?to - location)` predicate, indicating that the packages can move in one direction from `?from` warehouse down the line to `?to` (either warehouse or destination). A current location of the package is denoted in the `(at ?m - package ?r - location)` predicate.

The final destination of each package is specified in the `(address ?p - package ?d - destination)` predicate. The goal is to deliver all packages to their destinations, and the fact that the package was successfully delivered is denoted with the `(delivered ?p - package)` predicate.

The vehicles assigned to warehouses are specified with the static `(depo ?v - vehicle ?w - warehouse)` predicate. A vehicle is either in its warehouse `(at-depo ?v - vehicle)`, or delivering `(delivering ?v - vehicle)` packages. The delivery of the package takes two steps: the actual delivery and then the returning of the vehicle to its warehouse (see Listing 6.3).

Listing 6.3: An excerpt from the hierarchical logistics action definition for a car agent.

```
; the action describing delivering of a package
(:action send-car-one
    :parameters (?v - car ?from - warehouse
                 ?to - warehouse ?p1 - package)
    :precondition (and
        (depo ?v ?from)
        (at-depo ?v)
        (map ?from ?to)
        (at ?p1 ?from)
    )
    :effect (and
        (not (at ?p1 ?from))
        (not (at-depo ?v))
        (delivering ?v)
        (at ?p1 ?to)
    )
)
; the action describing returning of a car to its
   warehouse
(:action return
    :parameters (?v - vehicle)
    :precondition
        (delivering ?v)
    :effect (and
        (not (delivering ?v))
        (at-depo ?v)
    )
)
```

6.2.3 Problems

For this domain we designed a set of problems with various configurations of warehouses, vehicles, and packages (initial states of one of the simplest of these problems is showed in Figure 6.4) to see how different changes in the description of the problems affect the ability of planners to solve them.

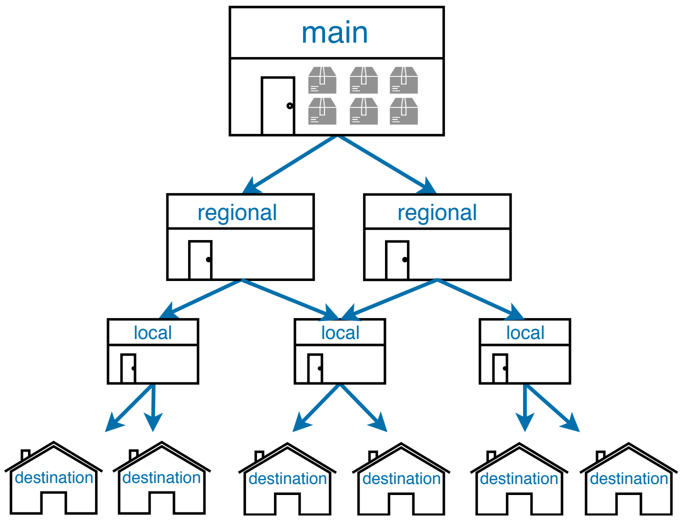

Figure 6.4: Illustration of a problem from the hierarchical logistics domain.

6.3 Manufacturing

The last domain puts emphasis on interaction between asymmetrical agents, another issue currently underrepresented in the set of benchmarks for multi-agent planners.

This domain models car manufacturing process starting with production of components and parts and finishing with putting all parts together on a final assembly line.

6.3.1 Description

In the manufacturing domain, the objective is to produce a certain number of cars from scratch. The process starts with suppliers who provide components to factories to produce car parts, such as the engine, interior and chassis. After that the parts are transferred to the assembly line, where the cars are assembled (see Figure 6.5).

6.3.2 Formalization

The objects in the implementation can be divided into three categories: (i) objects that represent the agents (`supplier`, `factory` and `assembly-line`), (ii) objects that represent locations (`supplier-location`, `factory-location` and `assembly-location`), and (iii) components of the future cars (`console`, `cylinder`, `metal-plate` and `extra`) plus ID numbers (`id`) assigned to cars to differentiate between them (see Listing 6.4).

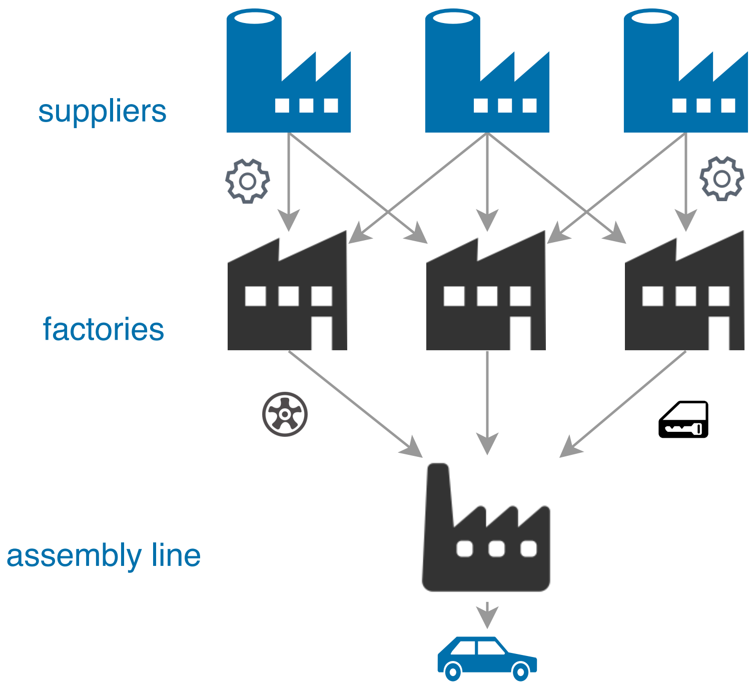

Figure 6.5: Illustration of the manufacturing domain.

Listing 6.4: An excerpt from the manufacturing domain definition.

```
(define (domain manufact)
(:requirements :strips :factored-privacy :typing)
(:types
    agent location component id - object
    supplier factory assembly-line - agent
    supplier-location factory-location
                    assembly-location - location
    console cylinder metal-plate extra - component
)
...
```

Each agent (`supplier`, `factory` and `assembly-line`) has its own location specified with the following private predicates:

- `(supplier-at ?s - supplier ?at - supplier-location)`,
- `(factory-at ?f - factory ?at - factory-location)`,
- `(assembly-line-at ?a - assembly-line ?at - assembly-location)`
  .

Transferring components and parts from one agent to another is realized through these locations as it would not be possible (with current specifications of MA-PDDL) to handle objects/predicates referring to components and parts directly through agents.

Suppliers are in control of the starting components of manufacturing. To note the fact that component `?c` belongs to a supplier located at supplier location `?at`, the predicate `(component-at-supplier ?c - component ?at - supplier-location)` is used. Suppliers ship components to factories with `:action ship-<x>` (see Listing 6.5), where `<x>` refers to a component (`console`, `cylinder`, `metal-plate`, `extra`). To ship component `<x>` to a factory (located at `?f`), it has to be specified that the factory uses `<x>` to produce parts `(uses-<x> ?f - factory-location)` and does not already have `<x>` `(no-<x>-at-factory ?f - factory-location)`. Predicate `(component-at-factory ?c - component ?at - factory-location)` indicates that component `?c` was shipped to a factory located at `?at`.

Listing 6.5: An excerpt from the manufacturing domain action definition for a supplier agent.

```
(:action ship-<x>
    :parameters (?s - supplier ?c - <x> ?from -
       supplier-location ?to - factory-location)
    :precondition (and
        (supplier-at ?s ?from)
        (uses-<x> ?to)
        (component-at-supplier ?c ?from)
        (no-<x>-at-factory ?to)
    )
    :effect (and
        (not (component-at-supplier ?c ?from))
        (component-at-factory ?c ?to)
        (not (no-<x>-at-factory ?to))
    )
)
```

Factories use components to produce car parts. A private predicate `(factory-produces-<y> ?f - factory)` indicates that factory `?f` can produce a part `<y>`, where `<y>` refers to a car part (engine, interior, chassis). Different car parts require different components, e.g. an engine is produced using a cylinder and a metal-plate (see Listing 6.6). Predicate `(no-part-at-factory ?f - factory-location)` indicates that the factory located at `?f` does not have any produced parts. To specify that the part `<y>` is at factory's location, the predicate `(<y>-at-factory ?f - factory-location)` is used. It was decided that for car parts it was more suitable to use predicates instead of objects as there was no need to differentiate between car parts of the same type (what mattered was the fact that a particular car part was produced) and it significantly simplified the implementation.

Listing 6.6: An excerpt from the manufacturing domain definition for a factory agent.

```
(:action produce-engine
    :parameters (?f - factory ?req1 - cylinder
          ?req2 - metal-plate ?at - factory-location)
    :precondition (and
        (no-part-at-factory ?at)
        (factory-produces-engine ?f)
        (component-at-factory ?req1 ?at)
        (component-at-factory ?req2 ?at)
        (factory-at ?f ?at)
    )
    :effect (and
        (not (component-at-factory ?req1 ?at))
        (not (component-at-factory ?req2 ?at))
        (not (no-part-at-factory ?at))
        (no-metal-plate-at-factory ?at)
        (no-cylinder-at-factory ?at)
        (engine-at-factory ?at)
    )
)
```

After that the parts are transferred to the assembly line, where they are used to assemble a car. Predicate `(<y>-at-assembly-line ?f - assembly-location)` indicates that `<y>` is at `?f` and `(no-<y>-at-assembly-line ?f - assembly-location)` indicates that the factory located at `?f` does not have part `<y>`. To assemble a car, the assembly line has to have in its disposition an engine, interior and chassis. To keep count of the number of cars produces, the ID numbers `(available ?id - id)` are used. To indicate that a car was produced and an ID was assigned to it, the predicate `(car-assembled-with-id ?id - id)` is used.

6.3.3 Problems

For this domain we designed a set of problems with various configurations of suppliers and factories, and numbers of cars to produce to see how different changes in the description of the problems affect the ability of planners to solve them.

6.4 Complexity of the problems

To speak about the complexity of multi-agent problems, there are two factors to consider:

- A problem-specific factor involving such parameters as the size and overall specifications of problems, the number of agents and their spatial distribution within a particular problem.
- A domain-specific factor, which includes the structure of relations between agents and uniformity (homogeneous vs. heterogeneous) of the agents.

Two intuitive parameters can also be adopted from [BD08]. The domain-specific measure of how much the agents have to coordinate with each other and a problem-specific measure of how much agents influence each other's actions.

The lanterns domain concerns homogeneous agents with a high level of coordination and no restrictions on interaction between them.

The hierarchical logistics domain concentrates on a hierarchical ordering of the semi-uniform agents according to their capabilities.

The manufacturing domain is mainly focused on heterogeneous agents, with each agent having a defined set of agents with which it can interact.

6.5 Summary

In this chapter we proposed novel domains, presented their implementation in MA-PDDL, and designed problems that will be used to experimentally compare selected existing multi-agent planners.

Chapter 7

Experimental evaluation

In this chapter we provide and evaluate the results of testing the planners from Chapter 5 on the domains and the problems presented in Chapter 6 to see how different algorithms and heuristics handle the problems of different complexity.

7.1 Setup & Evaluation

All planners were run on one machine (from the command line on Ubuntu). Each planner was given a 30-minute time limit and 1GB of memory limit per problem.

The evaluation consisted of three metrics: the total number of problems solved, an IPC quality score, and an IPC agile score adopted from [VCM18].

The IPC quality score estimates the quality of the plans produced by planners. For this metric, it only matters if a solution was found within 30 minutes and its cost. The quality score of a planner is calculated as follows. Let Q^* be the cost of the best found or optimal solution for the problem. Let Q be the cost of the solution found by the planner, $Q \geq Q^*$ (if lower costs are considered best). Higher sums of all quality ratios Q^*/Q are more desirable.

The IPC agile score allows the comparison of planning speed. In this metric, the quality of solutions is ignored and it only matters whether a solution to a problem was found within a 5-minute time limit or not and the time required to find it. It is calculated as follows. If the problem was solved within 1 second, the planner gets 1 score for the problem. If the problem was not solved within 5 minutes, the planner gets 0. For problems solved in the interval between 1 second and 5 minutes, the planner gets the score $1/(1 + T/T^*))$, where T^* is the minimum time required to solve the problem (by any planner). Higher sums of all scores are more desirable.

7.2 Results

Tables 7.1-7.3 show the specifications of individual problems and whether or not a particular planner was able to solve them with restrictions discussed above. For convenience, the parameters of the problems were colored from light blue to bright blue to indicate the gradation from lower to higher values of parameters.

Tables 7.4-7.6 show the costs of solutions to individual problems found by the planners.

As we can see from Table 7.1, Fast-Forward heuristics (MAPlan-FF, MADLA-FF, MADLA-PPsaFF) had no troubles with homogeneous agents, although as the individual parameters of the problems grow (and hence the level of coordination required to solve these problems), it becomes obvious that planners' capabilities do not go beyond trivial problems.

From Table 7.2, it can be seen that hierarchical structure of the problems (in this case the problems are extremely simple) is challenging for multi-agent planners. Further, a correlation can be traced between the number of packages in a problem (and hence the number of conditions for goal state to be reached) and how many planners were able to solve it, whereas the number of agents in this case does not play a big role.

The manufacturing problem is formulated in the way that it always takes the same number of actions to produce a car (Table 7.6 reflects this fact) and the challenge is to coordinate the actions between asymmetrical agents. Table 7.3 shows that the MADLA planner is extremely good at it and this can be attributed to its underling best-first search algorithm, as the variation of Fast-Forward heuristic in MAPlan did not even remotely produce such good results.

The graphs in Figures 7.1-7.3 show the number of solved problems over time for each domain. The majority of the problems were solved withing the 5 minute (300 second) time limit. There are also similarities noticed in the shape of the curves of the heuristics of the same planners, which can be explained by the same underlying algorithms used in the planners.

As can be seen from Tables 7.4-7.6, the results of the MAPlan with the Fast-Forward (MAPlan-FF) heuristic are very close to the results of optimal planning using the admissible LM-Cut (MAPlan-LM-Cut) heuristic. In contrast to this, the results of the MADLA planner in the case of the lanterns domain (with both projected Fast-Forward (MADLA-FF) and distributed Fast-Forward (MADLA-PPsaFF) heuristics) are very far from being reasonable, albeit that nevertheless they were obtained in a much shorter time (see Table 1.B in Appendix B) compared to MAPlan.

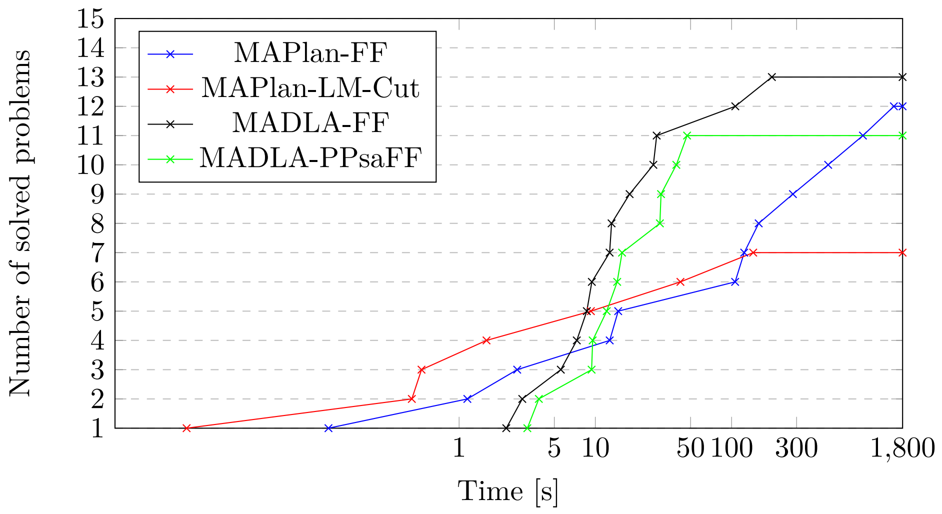

Figure 7.1: Number of solved problems over time for lanterns domain.

Finally, Table 7.7 shows the overall results of testing MAPlan and MADLA on the proposed domains. The table contains the metrics for the evaluation discussed above, with the total number of solved problems calculated from Tables 7.1-7.3, quality scores calculated from Tables 7.4-7.6, and agile scores calculated from Tables 1.B-3.B in Appendix B.

For the lanterns domain it can be said that MAPlan produces higher quality plans, while MADLA is better at planning time: this agrees with the overall results of the CODMAP competition [vKK15].

The MAPlan with Fast-Forward heuristic performed significantly better in terms of the quality of the plans and time required to solve the problems than other configurations in the hierarchical logistics domain. The manufacturing domain turned out to be challenging for MAPlan planner, while MADLA solved all the problems from this domain within the 5 minute time limit.

The MAPlan with LM-Cut heuristic did not outperform other planners in any category, which was expected from the only optimal configuration here tested.

On the whole, the proposed domain and problems showed potential for assessing multi-agent planners from different angles.

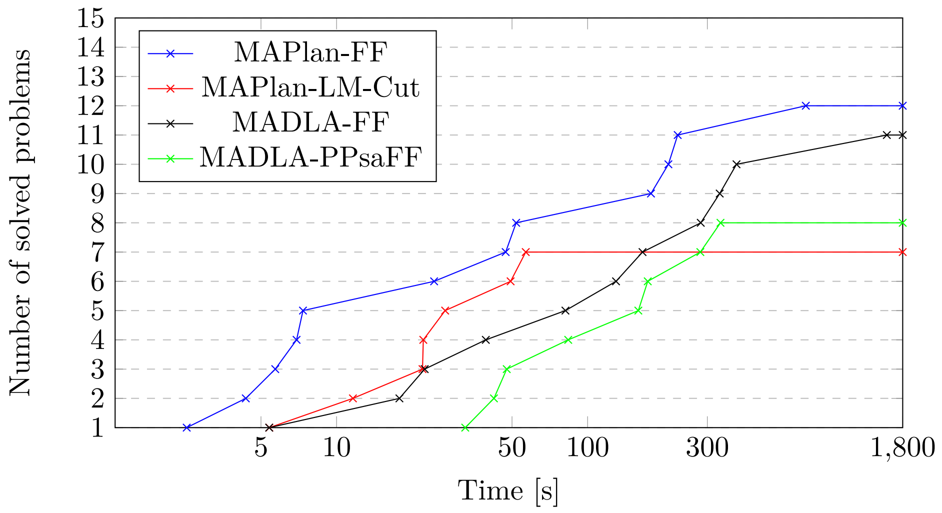

Figure 7.2: Number of solved problems over time for hierarchical logistics domain.

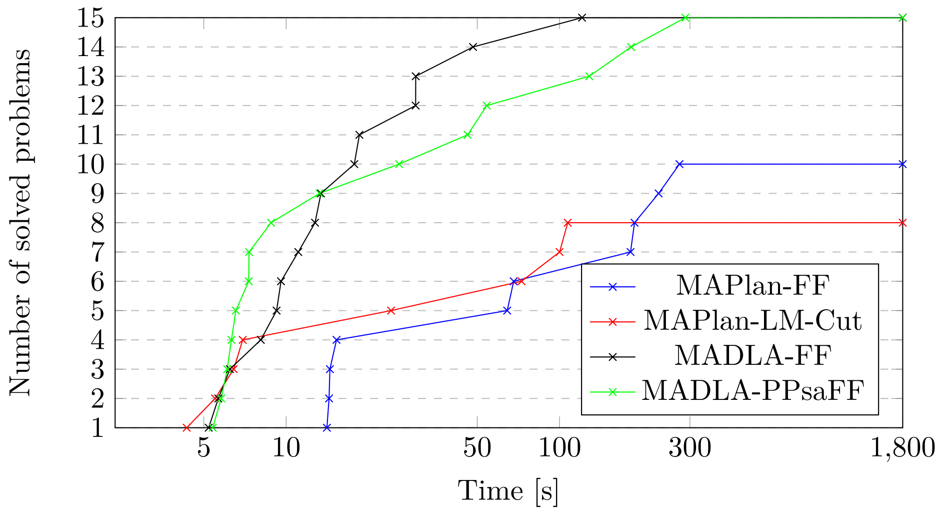

Figure 7.3: Number of solved problems over time for manufacturing domain.

Table 7.1: Coverage of the problems from the lanterns domain.

Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Num. of lanterns	3	2	4	6	5	3	4	2	4	6	3	5	3	3	5
Num. of switches	3	6	6	6	6	6	6	8	8	8	9	9	11	11	11
Num. of players	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4
MAPlan-FF	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	-	✓	✓	-
MAPlan-LM-Cut	✓	✓	✓	-	✓	✓	-	✓	-	-	✓	-	-	-	-
MADLA-FF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	-
MADLA-PPsaFF	✓	✓	✓	-	✓	✓	✓	-	✓	✓	✓	✓	-	✓	-

Table 7.2: Coverage of the problems from the hierarchical logistics domain.

Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Num. of cars	2	2	2	2	3	3	3	3	2	2	2	4	4	4	4
Num. of drones	3	3	3	3	4	4	4	4	6	3	3	3	3	5	5
Num. of packages	6	8	7	9	7	6	6	8	8	6	9	6	10	8	10
MAPlan-FF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	-	✓	-
MAPlan-LM-Cut	✓	-	✓	-	✓	✓	✓	-	-	✓	-	✓	-	-	-
MADLA-FF	✓	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	-	-	-
MADLA-PPsaFF	✓	✓	✓	-	✓	✓	✓	-	-	✓	✓	-	-	-	-

Table 7.3: Coverage of the problems from the manufacturing domain.

Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Num. of factories	1	2	1	1	2	2	1	3	2	3	3	3	2	1	1
Num. of suppliers	1	1	2	2	2	2	3	1	3	2	3	4	2	3	2
Cars to assemble	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3
MAPlan-FF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-
MAPlan-LM-Cut	✓	✓	✓	✓	-	-	✓	✓	✓	✓	-	-	-	-	-
MADLA-FF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MADLA-PPsaFF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 7.4: Plan costs of the problems from the lanterns domain.

Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MAPlan-FF	36	27	47	36	52	31	40	41	58		54		46	47	
MAPlan-LM-Cut	36	27	47		52	26		38			52				
MADLA-FF	42	47	99	431	194	57	118	58	101	221	141	148		82	
MADLA-PPsaFF	38	37	159		220	32	513		147	321	108	420		130	

Table 7.5: Plan costs of the problems from the hierarchical logistics domain

Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MAPlan-FF	18	24	22	28	22	17	16	24	25	18		21		26	-
MAPlan-LM-Cut	18		22		22	17	16			18		21			
MADLA-FF	22	30	26	34	26	22	21		26	21	34	31			
MADLA-PPsaFF	22	30	26		28	22	20			23	32				

Table 7.6: Plan costs of the problems from the manufacturing domain.

Problem	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MAPlan-FF	26	26	26	26	26	26	26	26	26	26					
MAPlan-LM-Cut	26	26	26	26			26	26	26	26					
MADLA-FF	26	26	26	26	26	26	26	26	26	26	26	26	39	39	39
MADLA-PPsaFF	26	26	26	26	26	26	26	26	26	26	26	26	39	39	39

Table 7.7: Coverage of individual domains.

	Lantens			Hierarch.logistics			Manufacturing		
	Solved Total	Quality Score	Agile Score	Solved Total	Quality Score	Agile Score	Solved Total	Quality Score	Agile Score
MAPlan-FF	12	11.73	5.51	12	12.00	10.68	10	10.00	5.69
MAPlan-LM-Cut	7	7.00	5.91	7	7.00	5.43	8	8.00	6.36
MADLA-FF	13	7.22	9.90	11	9.11	6.34	15	15.00	13.53
MADLA-PPsaFF	11	5.38	8.63	8	6.61	4.71	15	15.00	12.59

Chapter 8

Conclusion

The goal of the thesis was to provide novel domains for domain-independent privacy-preserving multi-agent planning.

The theoretical part of the thesis presented standardized tools for formalizing and implementing domains and problems in classical planning and expanded it for privacy-preserving multi-agent planning. The implementation and testing of selected multi-agent planners is realized in the practical part of the thesis.

Within this work, three new domains were proposed and implemented using MA-PDDL:

- the lanterns domain that demands high levels of coordination between planning agents,
- the hierarchical logistics domain that explores hierarchical structure of the problems,
- the manufacturing domain that challenges multi-agent planners to effectively work with asymmetrical agents.

An evaluation of the selected multi-agent privacy-preserving planners on the presented domains showed potential for using the domains for assessment of planners and how well they deal with the issues of different complexity.

The presented domains are to complement already existing multi-agent domains and to be used by multi-agent planning community for benchmarking, testing and possible future multi-agent planning competitions.

Appendices

Appendix A

Bibliography

[BD08] Ronen I. Brafman and Carmel Domshlak, *From One to Many: Planning for Loosely Coupled Multi-agent Systems*, Proceedings of the Eighteenth International Conference on International Conference on Automated Planning and Scheduling, ICAPS'08, AAAI Press, 2008, pp. 28–35.

[BFMS17] Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith (eds.), *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, AAAI Press, 2017.

[BG00] Blai Bonet and Hector Geffner, *Planning As Heuristic Search: New Results*, Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning (Berlin, Heidelberg), ECP '99, Springer-Verlag, 2000, pp. 360–372.

[cod15] *Main page of the Competition of Distributed and Multiagent Planners (CoDMAP)*, `http://agents.fel.cvut.cz/codmap/`, 2015.

[Dur01] Edmund H. Durfee, *Mutli-agents Systems and Applications*, Springer-Verlag New York, Inc., New York, NY, USA, 2001, pp. 118–149.

[Fiš16] Daniel Fišer, *MAPlan Planner*, `https://github.com/danfis/maplan`, 2016.

[FK18] Daniel Fišer and Antonín Komenda, *Concise Finite-Domain Representations for Factored MA-PDDL Planning Tasks*, Proceedings of the 10th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART, INSTICC, SciTePress, 2018, pp. 306–313.

[FN71] Richard E. Fikes and Nils J. Nilsson, *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*, Proceedings of the 2Nd International Joint Conference on Artificial Intelligence (San Francisco, CA, USA), IJCAI'71, Morgan Kaufmann Publishers Inc., 1971, pp. 608–620.

[FvK15] Damiel Fišer, Michal Štolba, and Antonín Komenda, *Maplan*, ICAPS Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15), 2015, pp. 8–10.

[GKW$^+$] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld, *PDDL - The Planning Domain Definition Language, 1998.*

[Has08] Patrik Haslum, *Writing Planning Domains and Problems in PDDL*, `http://users.cecs.anu.edu.au/~patrik/pddlman/writing.html`, 2008.

[HD09] Malte Helmert and Carmel Domshlak, *Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?*, Proceedings of the Nineteenth International Conference on International Conference on Automated Planning and Scheduling, ICAPS'09, AAAI Press, 2009, pp. 162–169.

[Kov] Daniel L. Kovacs, *A Multi-Agent Extension of PDDL3.1*, Proceedings of the 3rd Workshop on the International Planning Competition (IPC), 22nd International Conference on Automated Planning and Scheduling (ICAPS-2012), ICAPS, pp. 19–27.

[Kov15] Daniel L. Kovacs, *Complete BNF definition of MA-PDDL with privacy*, `http://agents.fel.cvut.cz/codmap/MA-PDDL-BNF-20150221.pdf`, 2015.

[KvK16] Antonín Komenda, Michal Štolba, and Daniel L. Kovacs, *The International Competition of Distributed and Multiagent Planners (CoDMAP)*, AI Magazine **37** (2016), no. 3, 109–115.

[NB14] Raz Nissim and Ronen Brafman, *Distributed Heuristic Forward Search for Multi-agent Planning*, J. Artif. Int. Res. **51** (2014), no. 1, 293–332.

[NGT04] Dana Nau, Malik Ghallab, and Paolo Traverso, *Automated Planning: Theory & Practice*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[RW17] Sean Leach Ryan Welham, Mike Gillespie, *TRYST on itch.io*, `https://welham14.itch.io/tryst`, 2017.

[TnOKv17] Alejandro Torreño, Eva Onaindia, Antonín Komenda, and Michal Štolba, *Cooperative Multi-Agent Planning: A Survey*, ACM Comput. Surv. **50** (2017), no. 6, 84:1–84:32.

[TnOS14] Alejandro Torreño, Eva Onaindia, and Óscar Sapena, *FMAP: Distributed Cooperative Multi-agent Planning*, Applied Intelligence **41** (2014), no. 2, 606–626.

[TvK] Jan Tožička, Michal Štolba, and Antonín Komenda, *The Limits of Strong Privacy Preserving Multi-Agent Planning*, Proceedings International Conference on Automated Planning and Scheduling, ICAPS. Association for the Advancement of Artificial Intelligence (AAAI), 2017. pp. 297-305.

[VCM18] Mauro Vallati, Lukás Chrpa, and Thomas Leo McCluskey, *What you always wanted to know about the deterministic part of the International Planning Competition (IPC) 2014 (but were too afraid to ask)*, Knowledge Eng. Review **33** (2018), e3.

[vK14] Michal Štolba and Antonín Komenda, *Relaxation Heuristics for Multiagent Planning*, Proceedings International Conference on Automated Planning and Scheduling, ICAPS, vol. 2014, 06 2014.

[vK15] Michal Štolba and Antonín Komenda, *MADLA: Planning with Distributed and Local Search*, ICAPS Proceedings of the Competition of Distributed and Multi-Agent Planners (CoDMAP-15), 2015, pp. 8–10.

[vK17] Michal Štolba and Antonín Komenda, *The MADLA Planner: Multi-Agent Planning by Combination of Distributed and Local Heuristic Search*, Artificial Intelligence, vol. 252, 08 2017.

[vKK15] Michal Štolba, Antonín Komenda, and Daniel L. Kovacs, *CoDMAP - Detailed Results*, `http://agents.fel.cvut.cz/codmap/results/presentation-RESULTS.pdf`, 2015.

[vKK16] Michal Štolba, Antonín Komenda, and Daniel L. Kovacs, *Competition of Distributed and Multiagent Planners (CoDMAP)*, Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, 2016, pp. 4343–4345.

[vO09] M. van Otterlo, *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for Adaptive Sequential Decision Making Under Uncertainty in First-order and Relational Domains*, EBSCO ebook academic collection, Ios Press, 2009.

[Š16] Michal Štolba, *MADLA Planner*, `https://github.com/stolba/MADLAPlanner`, 2016.

Appendix B

Tables

Problem	1	2	3	4	5
MAPlan-FF	1.15	0.11	14.72	1553.71	123.82
MAPlan-LM-Cut	0.53	0.01	9.28		144.12
MADLA-FF	2.91	2.22	5.58	17.85	13.16
MADLA-PPsaFF	3.17	3.85	12.12		14.47
Problem	6	7	8	9	10
MAPlan-FF	14.72	106.20	14.72	511.7	
MAPlan-LM-Cut	1.59		0.45		
MADLA-FF	8.66	12.73	7.31	9.43	106.49
MADLA-PPsaFF	9.56	29.78		15.71	39.32
Problem	11	12	13	14	15
MAPlan-FF	158.19		282.01	920.5	
MAPlan-LM-Cut	42.14				
MADLA-FF	26.66	197.64		28.21	
MADLA-PPsaFF	9.39	47.15		30.36	

Table B.1: Time required to solve individual problems from the lanterns domain.

Problem	1	2	3	4	5
MAPlan-FF	6.93	178.87	24.47	740.18	51.94
MAPlan-LM-Cut	11.62		22.16		49.36
MADLA-FF	39.32	81.61	129.8	1556.25	282.37
MADLA-PPsaFF	32.56	83.73	338.4		173.64
Problem	6	7	8	9	10
MAPlan-FF	7.35	5.7	47.17	209.84	4.35
MAPlan-LM-Cut	22.00	27.08			5.40
MADLA-FF	165.54	335.97		392.00	5.39
MADLA-PPsaFF	159.27	281.33			47.63
Problem	11	12	13	14	15
MAPlan-FF		2.53		228.53	
MAPlan-LM-Cut		56.71			
MADLA-FF	17.8	22.44			
MADLA-PPsaFF	42.36				

Table B.2: Time required to solve individual problems from the logistics domain.

Problem	1	2	3	4	5
MAPlan-FF	14.1	181.9	15.29	14.36	188.14
MAPlan-LM-Cut	4.33	72.65	6.94	6.43	
MADLA-FF	18.54	5.65	9.58	17.76	6.23
MADLA-PPsaFF	5.39	7.32	25.9	13.24	6.09
Problem	6	7	8	9	10
MAPlan-FF	68.00	14.46	230.45	64.36	274.67
MAPlan-LM-Cut		5.5	100.05	24.2	107.17
MADLA-FF	12.76	8.08	5.21	29.79	13.41
MADLA-PPsaFF	46.11	6.55	5.79	54.21	6.31
Problem	11	12	13	14	15
MAPlan-FF					
MAPlan-LM-Cut					
MADLA-FF	9.23	11.08	29.77	48.27	120.91
MADLA-PPsaFF	8.81	7.3	288.55	182.84	128.42

Table B.3: Time required to solve individual problems from the manufacturing domain.